Sub A1

BRANCH PREDICTION METHOD USING ADDRESS TRACE

This application relies for priority upon Korean Patent Application No. 1999-50627, filed on November 15, 1999, the contents of which are herein incorporated by reference in their entirety.

Background of the Invention

1. Field of the Invention

Sub A2

The present invention relates to a branch prediction method and, more particularly, to a branch prediction method using an address trace.

2. Description of the Related Art

Fig. 1 shows fourth generation microarchitecture, which is cited from Figure 1 of a paper entitled "Trace Processors: Moving to Fourth Generation Microarchitecture," IEEE Computer, pp. 68-74, September 1997, by James E. Smith & Sriram Vajapeyam. In Fig. 1, (a) is the first generation microarchitecture serial processors which began in the 1940s with the first electronic digital computers and ended in the early 1960s. The serial processors fetch and execute each instruction before going to the next. Diagram (b) is the second generation microarchitecture using pipelining. Such architectures were the norm for high-performance processing until the late 1980s. Diagrams (c) and (d) are the third and fourth generation microarchitectures that are characterized by superscalar processors. The third and fourth generation architectures first appeared in commercially available processors in the late 1980s.

As shown in Fig. 1, high-performance processors of the next generation will be composed of multiple superscalar pipelines, with some higher level control that dispatches groups of instructions to the individual superscalar pipes. The superscalar processors, which can patch a group of instructions to a cache block at the same time and process the instructions in parallel, are afflicted with performance degradation resulting from pipeline stall and abort of missed instruction rather than prediction error. Therefore, correct branch prediction is very significant to the superscalar processors.

As mentioned above, a critical technology to achieve high performance of superscalar processors is to optimally use pipelines. The most significant design factor is a branch prediction method. A branch predictor operates to predict the outcome of a branch instruction before performing a condition check of the branch instruction based on a predetermined branch prediction approach. A central processing unit (CPU) then fetches the next instruction according to the predicted result. A pipeline technique is adopted to solve a pipeline stall phenomenon that causes performance degradation of a CPU. However, when a branch prediction is missed, many instructions from the incorrect code section may be in various stages of processing in the instruction execution pipeline.

On encountering such a misprediction, instructions following the mispredicted conditional branch instruction in the pipeline are flushed, and instructions from the other, correct code section are fetched. Flushing the pipeline creates bubbles or gaps in the pipeline. Several clock cycles may be required before the next useful instruction completes execution, and before the instruction execution pipeline produces useful output. Such incorrect guesses cause the pipeline to stall until it is refilled with valid instructions, this delay is called the mispredicted branch penalty.

Generally, a processor with 5-step pipelines has a branch penalty of two cycles. For example, in 4-way superscalar design, 8 instructions are subject to loss. If a pipeline is expanded, more instructions are subject to loss and the branch penalty increases. Since programs generally branch in every 4 or 6 instructions, misprediction causes acute performance degradation in a deep pipeline design.

There have been efforts to reduce the above-mentioned branch penalty. Recently, a trace processor using a trace cache has been applied. The trace processor is disclosed in the paper "Trace Processors: Moving to Fourth Generation Microarchitecture," by James E. Smith & Sriram Vajapeyam.

Fig. 2A shows a dynamic sequence of basic blocks embedded in a conventional instruction cache 21. Fig. 2B shows a conventional trace cache 22. Referring now to Fig. 2A, arrows indicate a branch “taken” (jumping to target code section). In the instruction cache 21, even multiple branch predictions created at every cycle require 4 cycles to fetch instructions in basic blocks "ABCDE" because instructions are stored in discontinuous caches.

Accordingly, some have proposed a specific instruction cache to capture long dynamic instruction sequences. Each line of the specific instruction cache stores a snapshot or trace of a dynamic instruction stream, as shown in Fig. 2B. This cache is referred to as a trace cache 22, which is disclosed in a paper entitled "Expansion Caches for Superscalar Processors," Technical Report CSL-TR-94-630, Stanford Univ., June 1994, by J. Johnson; U. S. Patent No. 5,381,533 entitled "Dynamic Flow Instruction Cache Memory Organized Around Trace Segments Independent Of Virtual Address Line," issued on January 1995 to A. Peleg & U. Weiser; and a paper entitled "Trace Cache: A Low Latency Approach To High Bandwidth Instruction Fetching," Pro. 29th Int'l Symp. Microarchitecture, pp. 24-34, December 1996, by E. Rotenberg, S. Bennett, & J. Smith.

Furthermore, the trace cache 22 is disclosed in a paper entitled "Improving Trace Cache Effectiveness with Branch Promotion and Trace Packing," Proc. 25th Int'l Symp. Computer Architecture, pp. 262-271, June 1998, by Sanjay Jeram Patel, Marius Evers, and Yale N. Patt; a paper entitled "Evaluation of Design Options for the Trace Cache Fetch Mechanism," IEEE TRANSACTION ON COMPUTER, Vol. 48, No. 2, pp. 193-204, February 1999, by Sanjay Jeram Patel, Daniel Holmes Friendly & Yale N. Patt; and a paper entitled "A Trace Cache Microarchitecture and Evaluation," IEEE TRANSACTION ON COMPUTER, Vol. 48, No. 2, pp. 111-120, February 1999, by Eric Rotenberg, Steve Bennett, and James E. Smith.

Sub A3

A dynamic sequence, which is identical to the discontinuous blocks in the instruction cache 21 shown in Fig. 2A, is continuous in the trace cache 22 shown in Fig. 2B. Therefore, instructions stored in the trace cache 22 can sequentially be executed without repeated branch to an address including an instruction according to a conventionally programmed routine. This makes it possible to prevent a branch penalty which occurs in conventional prediction techniques. And, instructions stored in discontinuous positions of the instruction cache 21 are continuously stored in the trace

A3 cache to carry out improved parallel processing.

Because it stores an instruction itself, the trace cache 22 requires decoding to an address corresponding to the instruction. And because the trace cache 22 repeatedly stores even repetitively executed instructions according to their execution order, a chip size of the trace cache 22 increases too much. In order to have enough size to store all instructions, the trace cache 22 inevitably increases in chip size and manufacturing cost.

Summary of the Invention

Therefore, it is an object of the present invention to provide a branch prediction approach using a trace cache, which can shorten address decoding time and decrease chip size and manufacturing cost.

Sub A4 According to the aspect of the present invention, a branch prediction method uses a trace cache. If a routine composed of unrepeated instructions is to be executed, an address corresponding to each instruction in the trace cache according to an order of executed instructions is stored. If a routine composed of repeated instructions is carried out, a routine start address, a routine end address, current access times of the routine, and total access times of the routine are counted and stored.

Sub A5 If the routine composed of the repeated instructions is carried out, the trace cache includes loop counters for counting the current access times and the total current access times. If values of the loop counters are identical to each other, a start address that will be subsequent to the routine is addressed. If the branch prediction is missed, the loop counter is recomposed using the latest updated loop count value.

Brief Description of the Drawings

The foregoing and other objects, features and advantages of the invention will be apparent from the more particular description of preferred embodiments of the invention, as illustrated in the accompanying drawings in which like reference characters refer to the same parts throughout the different views. The drawings are not necessarily to scale, emphasis instead being placed upon illustrating the principles of the invention.

Sub A6 Fig. 1 contains schematic functional block diagrams which illustrate various microarchitectures including fourth-generation microarchitectures.

Sub A7 Fig. 2A is a schematic diagram which shows a dynamic sequence of basic blocks stored in a conventional instruction cache.

Sub A8 Fig. 2B is a schematic diagram which illustrates a conventional trace cache.

Sub A9 Fig. 3 is a schematic diagram which illustrates one example of a repetitive instruction pattern.

Fig. 4 is a schematic diagram which illustrates a pattern where instructions shown in Fig. 3 are stored in a trace cache according to prior art.

Fig. 5 is a schematic diagram which illustrates a structure of an address trace cache according to the present invention.

Fig. 6 is a schematic diagram which illustrates a pattern where instructions shown in Fig. 3 are stored in a trace cache according to the present invention.

Ins A10

Description of Preferred Embodiments of the Invention

A new and improved trace cache stores an address trace itself corresponding to an instruction with a decoded form, thus shortening address decoding time for each instruction. The new and improved trace cache uses a small amount of trace cache memory in storing an address trace to a repetitively executed routine, thus decreasing chip size and manufacturing cost.

Fig. 3 illustrates one example of a repetitive instruction pattern. In a routine 1, operations A and B are repeated 30 times. When the routine 1 is finished, a routine 2 is carried out wherein operations C, D, and E are sequentially repeated 20 times. When the routine 2 is finished, a routine 3 is carried out wherein operations F and G are repeated 40 times.

Assuming that, for example, the routines shown in Fig. 3 are carried out, Fig. 4 shows instructions that are stored in a trace cache 22 according to prior art. Referring now to Fig. 4, the trace cache 22 stores instructions based upon their execution order irrespective of the fact that executed instructions are repeatedly carried out. Accordingly, the trace cache 22 requires a data storing area for storing 60 instructions (2 instructions x 30 times repetition = 60) for routine 1, a data storing area for storing 60 instructions (3 instructions x 20 times repetition = 60) for routine 2, and a data storing area for storing 80 instructions (2 instructions x 40 times repetition = 80) for routine 3. That is, a total of 200 data storing areas are required for storing instructions for routines 1, 2, and 3 shown in Fig. 3. If 32 bits are required for storing each of the instructions, a data storing area of total 6400 bits (i.e., 800 bytes) is required for routines 1, 2, and 3.

Sub A11

In Fig. 5, in accordance with the invention, an address trace cache 220 is composed of a start address for storing an address where each routine is started, an end address for an address where each routine is finished, an access current loop counter for counting access times of a corresponding routine, and an old access loop counter for indicating total access times of the routine.

Sub A12

For example, if the information of instructions executed in routine 1 is indicated the address trace cache 220, the start address and the end address of routine 1 are stored in the trace cache 220. Then, current access time of routine 1 is stored in the current access loop counter while total access times (e.g., 30 times) of routine 1 is stored in the old access loop counter. As access of the routine is repeated, a value of the current access loop counter is increased. If the value of the current access loop counter is identical to that of the old access loop counter, routine 1 is finished and a start address of routine 2 is stored as a next fetch point (NFP).

As shown in Fig. 3 and Fig. 5, if routine 1 through 3 are successively carried out, they can be stored in the address trace cache 220 by the above-mentioned manner, respectively. If routines 1 through 3 are not repeatedly carried out, the address of each instruction composing routines 1 through 3 is sequentially written therein. When branch prediction is missed, the loop counter is recomposed with the latest updated loop count value.

Sub A13

Referring now to Fig. 6, if, for example, a routine 1 shown in Fig. 3 is stored in the address trace cache 220, an address of an initially executed instruction A is stored into a start address of routine 1 while an address of a finally executed instruction B is stored into

A13

an end address thereof. Since total repetition times of the routine 1 is 30, an old access loop counter is stored as 30. Whenever routine 1 is repeatedly carried out, a value of a current access loop counter increases by 1. In the same manner, information of routines 2 and 3 is stored in the address cache 220.

Sub A14

As shown in Fig. 6, since the address cache 220 is composed of an address where each routine is started, an address where each routine is finished, a current access loop counter, and an old access loop counter, only four data storing areas are required to store information of a repeated routine. Therefore, total 12 data storing areas are required to store routines 1 through 3 in the address trace cache 220. In this case, if 32 bits are utilized to store each piece of information, a total of 384 bits (i.e., 48 bytes) are required to store routines 1 through 3. This is smaller by about 16.7 times than a data storing area utilized in a conventional trace cache.

Int A15

Such an effect may be significant, as the number of repeated instructions in a routine becomes large or the number of repetitions of instructions rises. A trace cache of this invention utilizes a data storing area that is considerably reduced in comparison with a conventional trace cache, resulting in decreased chip size and manufacturing unit price. Moreover, the trace cache stores an address trace itself to each instruction with a decoded form, reducing an address decoding time of the instruction.

While this invention has been particularly shown and described with references to preferred embodiments thereof, it will be understood by those skilled in the art that various changes in form and detail may be made therein without departing from the spirit and scope of the invention as defined by the appended claims.